GRAPH ANALYTICS LIBRARIES COMPARISON

A Thesis
Presented to
The Academic Faculty

by

Yili Hui

In Partial Fulfillment
of the Requirements for the Degree
Computer Science in the
School of College of Computing, Georgia Institute of Technology

Georgia Institute of Technology
May 2019

GRAPH ANALYTICS LIBRARIES COMPARISON

Approved by:

Dr. Polo Chau, Advisor
School of Computational Science & Engineering
Georgia Institute of Technology

Dr. Richard Peng
School of College of Computing
Georgia Institute of Technology

ACKNOWLEDGEMENTS

I wish to thank William Xia and Swati Mardia for giving comments and proofreading my proposal and thesis.

TABLE OF CONTENTS

LIST OF TABLES

ABSTRACT

In this undergraduate research capstone project, we compare three popular graph analytics libraries --- igraph, NetworkX and SNAP --- by using four large real-world graphs with up to 69 million edges. We then examine the runtimes of three graph generation methods and five graph algorithms. All experiments were run on Partnership for an Advanced Computing Environment (PACE) servers at Georgia Tech.

CHAPTER 1

INTRODUCTION

Understanding large graphs is imperative for researchers and scientists working in a variety of disciplines. Over the years, a number of graph analytics libraries have been developed. Prominent examples include igraph, NetworkX and Stanford Network Analysis Project (SNAP). In this undergraduate research capstone project, we compare the performance of igraph, NetworkX and SNAP by analyzing a number of algorithms' runtimes. While there is significant amount of prior work on studying graph algorithms and graph visualization tools, there is much less work conducted on comparing graph analytics libraries. We believe our work can help researchers and practitioners make informed decisions in choosing graph libraries for their work.

Using prior research [4] as a blueprint, we evaluate these three libraries using four large real-world graphs, with up to 69 million edges. The graph operations we have selected include: PageRank, clustering coefficient, weakly-connected components, 3-core and testing edge existence. These algorithms are important for understanding graphs. PageRank measures importance of nodes. Clustering coefficient and weakly-connected components reveal structures in graphs, and Testing edge existence checks for connections between nodes. Additionally, we select these algorithms based on prior work [4], which would allow us to more easily discuss and compare our results with prior research.

Our experiments have three parts: First, we generate large graphs using Erdős–Rényi, Albert-Barabasi preferential attachment and Watts-Strogatz small-world graph generators and record their generation runtimes. Second, we calculate the time needed to load

datasets. Third, we run the five graph algorithms mentioned above. Chapter 2 discusses the details of the datasets. Chapter 3 explains how the experiments are conducted. Chapter 4 discusses the results. Chapter 5 gives the conclusion.

To ensure reproducibility of our work, we have open-sourced our code on GitHub at https://github.com/Yili0616/graph-analytics-comparison.

CHAPTER 2

DATASETS

For our experiment, we use four real-world graph datasets downloaded from the, Stanford Large Network Dataset Collection [3] (http://snap.stanford.edu/). Table 1 describes these graphs' basic information: node counts, edge counts, and edge directedness. We pick one small (under 10 M edges), two medium (from 10M to 50M) and one large datasets in order to test scalability of igraph, NetworkX and SNAP. They are wiki-Talk, cit-Patents, soc-Pokec and soc-LiveJournal1.

The wiki-Talk graph is a Wikipedia talk network. On Wikipedia, each registered user has a talk page. Nodes are Wikipedia users. A directed edge connects user u to user v, if u has edited v's talk page at least once. The cit-Patents graph is a citation network for US Patents. Nodes represent patents, and edges are citations among patents. The soc-Pokec and socLiveJournal1 graphs are both on-line social networks. Nodes are users. Edges represents friendships between users.

Graph	**Nodes**	**Edges**	**Directed?**
wiki-Talk	2,394,385	5,021,410	Directed
cit-Patents	3,774,768	16,518,948	Directed
soc-Pokec	1,632,803	30,622,564	Directed
soc-LiveJournal1	4,847,571	68,993,773	Directed

Table 1: Four real-world graphs, from SNAP, used in experiments in this work. Graphs ordered by their sizes.

Nodes	**Edge Created by Each Node**
1M	10

Table 2: Parameters of Albert-Barabasi Preferential Attachment

Nodes	**Neighbors**	**Rewiring Probability**
100k	200	0.5
1M	200	0.5

Table 3: Parameters of Watts-Strogatz Small-World

CHAPTER 3

EVALUATION METHODS

Based on the SNAP paper [4], we measure the time to generate G(1M, 10M), G(1M, 100M) and G(10M, 100M) on the Erdős–Rényi random graph generator. As mentioned above, we also add two com-mon random graph generators to the experiments: Albert-Barabasi preferential attachment and Watts-Strogatz Small-World. Table 2 and Table 3 are summaries of parameters we used to generate those two types of graphs. In addition, we measure the loading and compute time of all graph analytics algorithms mentioned above for datasets in chapter 2. We remove self-loops from graphs for consistency since NetworkX does not implement 3-cores for graphs with self-loops. Every experiment is run independently for five iterations. Averages of runtimes are taken. All experiments are run on sake node (rich133-g24-17-r) from PACE at Georgia Tech. It has dual Intel Xeon E5-2680 v3 processor with128GB RAM and 8TB storage

CHAPTER 4

EVALUATION RESULTS

The codes for testing igraph, NetworkX and SNAP are written in R, Python and C++, respectively. Different programming languagesand data structures lead to different performance

4.1 RANDOM GRAPH GENERATORS

As stated in evaluation methods, we generate random graphs from Erdős–Rényi, Albert-Barabasi Preferential Attachment and Watts-Strogatz Small-World graph generators. Table 4, Table 5 and Table 6 are runtimes for each of them. igraph runs fastest for Erdős–Rényi, while SNAP is the best for Watts-Strogatz Small-World. According to the data, execution time grows linearly with the number of nodes and edges. One exception for Erdős–Rényi on SNAP is that increasing node number decreases execution time. The reason is that when generating random edges, SNAP needs to check whether the edge exists. It is easier to check the existence of edges in a sparse graph than in a dense graph. As mentioned above, adjacency nodes are stored in vectors. On average, if the graph is denser, there are more elements in a vector. Checking edge existence will take more time. There are two algorithms for Erdős–Rényi on NetworkX. The faster one with $O(m+n)$ is used instead of slower one with $O(n^2)$. n is the number of nodes and m is the number of expected edges. Table 5 and Table 6 are the results for Albert-Barabasi Preferential Attachment and Watts-Strogatz Small-World random graph generators. The runtimes of igraph and SNAP are very close except for 1M nodes Watts-Strogatz Small-World graph - SNAP is 60 seconds faster than igraph.

Graph Size		Execution Time (seconds)		
Nodes	**Edges**	**igraph**	**NetworkX**	**SNAP**
1M	10M	**5.5**	44.4	8.6
1M	100M	**54.3**	428.5	178.0
10M	100M	**63.3**	507.7	102.0

Table 4: Execution Time (seconds) for Erdős–Rényi

	Execution Time (seconds)		
Nodes	**igraph**	**NetworkX**	**SNAP**
1M	6.5	58.3	**5.5**

Table 5: Execution Time (seconds) for Albert-Barabasi Preferential Attachment

	Execution Time (seconds)		
Nodes	**igraph**	**NetworkX**	**SNAP**
100k	24.8	44.5	**20.9**
1M	291.1	481.2	**235.6**

Table 6: Execution Time (seconds) for Watts-Strogatz Small-World

4.2 LOADING GRAPHS

According to Table 7, SNAP has advantages over loading datasets. It is 10 times faster than igraph and 20 times faster than NetworkX. The reason for this discrepancy might be that vectors for igraph are hard to resize and implementation of hash table in NetworkX might not be as efficient as the one in SNAP. Measuring execution time for loading is imperative because loading time might take longer than some graph analytics algorithms execution time.

Graph	igraph	NetworkX	SNAP
wiki-Talk	26.8	60.8	**3.7**
cit-Patents	113.1	171.8	**11.6**
soc-Pokec	128.4	280.8	**19.8**
soc-LiveJournal1	309.6	701.1	**39.8**

Table 7: Execution Time (seconds) for Loading Datasets

4.3 GRAPH ANALYTICS

In this section, we compare execution time for PageRank, clustering coefficient, weakly-connected component, 3-core and testing edge existence.

Table 8 shows the result of page rank. We choose maximum number of iterations to be 10 and convergence difference to be 10^{-4}. There is less than 10 seconds difference between igraph and SNAP even if the sizes of graph tested change from 5M to 69M.

When calculating clustering coefficient in Table 9, we could see that igraph is 5 times to 10 times faster than SNAP. From 5M to 69M graphs, the runtimes for igraph increases 15 times while the runtimes for SNAP increases 7 times. It indicates that the runtimes for SNAP grows much slower than the runtimes for igraph. Thus, if the graphs are relatively small, igraph will be the best choice. However, if it comes to relatively large graph, SNAP can be optimal for the growth rate of runtime is low.

In Table 10, computation times for igraph and SNAP are similar. In general, efficient algorithm for finding weakly-connected components runs in linear time. Therefore, igraph, NetworkX and SNAP can discover weakly-connected components in under 10 seconds on most graphs.

Data in Table 11 shows that finding 3-core for graphs is indeed fast. However, NetworkX takes much longer than igraph and SNAP even if it implements 3-core in O(m). In fact, for page rank, clustering coefficient and 3-core, NetworkX runs much slower. The reason is that NetworkX is written in pure python code which is an

interpreted language. Generally, it takes longer time to execute code in Python comparing with code in C++. Even if we write code for igraph in R, igraph calls function from igraph C library.

In Table 12, we observe that testing edge existence takes only a few seconds. The way to test edge existence is that we randomly generate 20k node IDs with replacement, then we test whether there exists an edge between each consecutive pair of random nodes. In fact, time for testing edge existence 10k times is so small that a computer could not differentiate between start time and end time. Thus, it takes 0.00 seconds to test edge existence 10k times. The time difference is ignored.

Graph	**igraph**	**NetworkX**	**SNAP**
wiki-Talk	**1.3**	143.0	1.5
cit-Patents	**2.7**	375.2	7.3
soc-Pokec	**1.5**	612.3	3.9
soc-LiveJournal1	**4.6**	1589.1	10.1

Table 8: Execution Time (seconds) for PageRank

Graph	**igraph**	**NetworkX**	**SNAP**
wiki-Talk	**1.8**	3754.2	14.1
cit-Patents	**5.6**	403.7	15.6
soc-Pokec	**10.9**	1069.3	40.4
soc-LiveJournal1	**25.4**	3186.0	98.6

Table 9: Execution Time (seconds) for Clustering Coefficient

Graph	igraph	NetworkX	SNAP
wiki-Talk	**0.5**	5.0	0.7
cit-Patents	**2.1**	17.4	2.4
soc-Pokec	1.9	17.1	**1.7**
soc-LiveJournal1	4.8	37.2	**4.7**

Table 10: Execution Time (seconds) for Weakly-Connected Components

Graph	igraph	NetworkX	SNAP
wiki-Talk	0.5	87.3	**0.1**
cit-Patents	2.8	370.9	**0.2**
soc-Pokec	2.9	495.2	**0.1**
soc-LiveJournal1	8.0	1156.8	**0.3**

Table 11: Execution Time (seconds) for 3-Core

Graph	igraph	NetworkX	SNAP
wiki-Talk	0.72	**0.02**	**0.01**
cit-Patents	0.78	**0.01**	**0.01**
soc-Pokec	0.67	0.02	**0.00**
soc-LiveJournal1	0.78	0.03	**0.00**

Table 12: Execution Time (seconds) for Testing Edge Existence (10k)

4.4 OVERALL COMMENTS, AND COMPARISON WITH PRIOR WORK

SNAP and igraph are very stable in terms of runtimes. Usually, there is less than a one-second difference between the five trials that we run for each graph operation. NetworkX's runtimes, however, might differ by 10 times or more, likely due to the hash tables (and their inherent randomness) used in its data structures. We have observed that our runtimes are typically about half of those reported in prior work [4], which is likely

due to advances in hardware, since our hardware is more recent, and has higher CPU clock speed and bus speed. However, there is one exception. igraph has a much slower runtime for testing edge existence. It is possible that specific implementation such as a for loop in R is slower than the implementation in python and C++. We believe the runtime differences can be attributed to the different data structures used to store graphs in the libraries. Igraph [1] stores nodes and edges in vectors. In contrast, NetworkX [2] uses a 3D hash table to store nodes and adjacency lists. SNAP uses a combination of storage strategies. It stores nodes in an undirected graph in a hash table, and for each node, SNAP stores its adjacent nodes in a vector (or two vectors, in the case of a directed graph, one vector for in-neighbors, another for out-neighbors).

CHAPTER 5

DISCUSSION

From a usability perspective, igraph (in R) and NetworkX (in Python)may be easier to use, because both libraries can be readily importedinto users' program, and used in their workflows. SNAP, on theother hand, may have a steeper initial learning curve due to the lackof support for scripting interactive debugging capabilities providedby R and Python; SNAP programs (C++) would need to be compiledbefore users can test their code's correctness.

REFERENCES

[1] Gabor Csardi and Tamas Nepusz. 2006. The igraph software package for complex network research. Inter Journal, Complex Systems1695, 5 (2006), 1–9 [2] BROWN, G. L. and ROSHKO, A. “The effect of names in full upper case in numerical references,” J. Fluid Mech., vol. 26, pp. 225–236, 1966.

[2] Aric Hagberg, Pieter Swart, and Daniel S Chult. 2008. Exploring network structure, dynamics, and function using NetworkX. Technical Report. Los Alamos National Lab.(LANL), Los Alamos, NM (United States).

[3] Jure Leskovec and Andrej Krevl. 2015.{SNAP Datasets}:{Stanford}Large Network Dataset Collection. (2015).

[4] Jure Leskovec and Rok Sosič. 2016. Snap: A general-purpose network analysis and graph-mining library. ACM Transactions on Intelligent Systems and Technology (TIST)8, 1 (2016), 1.